AFRL-VS-PS-
TR-2007-1014

PCA RAW FABRIC: Architectural Prototyping, Demonstration and Evaluation

Prof. Saman Amarasinghe and Prof. Anant Agarwal

MIT-CSAIL
Stata Center, G32-782
32 Vassar Street
Cambridge, MA 02139

15 February 2007

Final Report

AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory/VS Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-VS-PS-TR-2007-1014 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

//signed//	//signed//
ANGELO BONAVITA, Capt, USAF Program Manager	JOHN P. BEAUCHEMIN, Lt Col, USAF Deputy Chief, Spacecraft Technology Division Space Vehicles Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15/02/2007	Final Report	10/03/2003 to 15/02/2007

4. TITLE AND SUBTITLE
PCA RAW FABRIC: Architectural Prototyping, Demonstration and Evaluation

5a. CONTRACT NUMBER
F29601-03-2-0065

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62712E

6. AUTHOR(S)
Prof. Saman Amarasinghe and Prof. Anant Agarwal

5d. PROJECT NUMBER
DARP

5e. TASK NUMBER
SC

5f. WORK UNIT NUMBER
AJ

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT-CSAIL
Stata Center, G32-782
32 Vassar Street
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Space Vehicles Directorate
3550 Aberdeen Ave., SE
Kirtland AFB, NM 87117-5776

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/VSSE

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-VS-PS-TR-2007-1014

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited. (Clearance #VS07-0266)

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report is the final technical report for the project ``PCA RAW Fabric: Architectural Prototyping, Demonstration, and Evaluation." This project successfully built and demonstrated a Raw fabric system containing 4 Raw chips, with 3 streaming fabric boards. In addition to the Raw fabric hardware, the project also developed and demonstrated StreamIt: a language and compiler specifically designed for embedded, high performance stream computing on PCA architectures such as Raw. The project also implemented, analyzed and distributed a new PCA benchmark suite called Versabench and a PCA performance metric called Versatility.

15. SUBJECT TERMS
Versabench, Parallel Programming, Polymorphic, Reconfigurable, Thread Programming, Stream Programming

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Capt Angelo Bonavita
Unclassified	Unclassified	Unclassified	SAR	42	19b. TELEPHONE NUMBER *(include area code)* (505) 846-0841

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

Contents

List of Figures

List of Tables

1 Abstract

This report is the final technical report for the project "PCA RAW Fabric: Architectural Prototyping, Demonstration, and Evaluation."

In this project, we successfully built and demonstrated a Raw fabric system containing 4 Raw chips, with 3 streaming fabric boards. The system is shown in the figure below. In addition to the Raw fabric hardware, we also developed and demonstrated StreamIt: a language and compiler specifically designed for embedded, high performance stream computing on PCA architectures such as Raw. Our project also implemented, analyzed and distributed a new PCA benchmark suite called Versabench and a PCA performance metric called Versatility (see http://cag.csail.mit.edu/versabench).

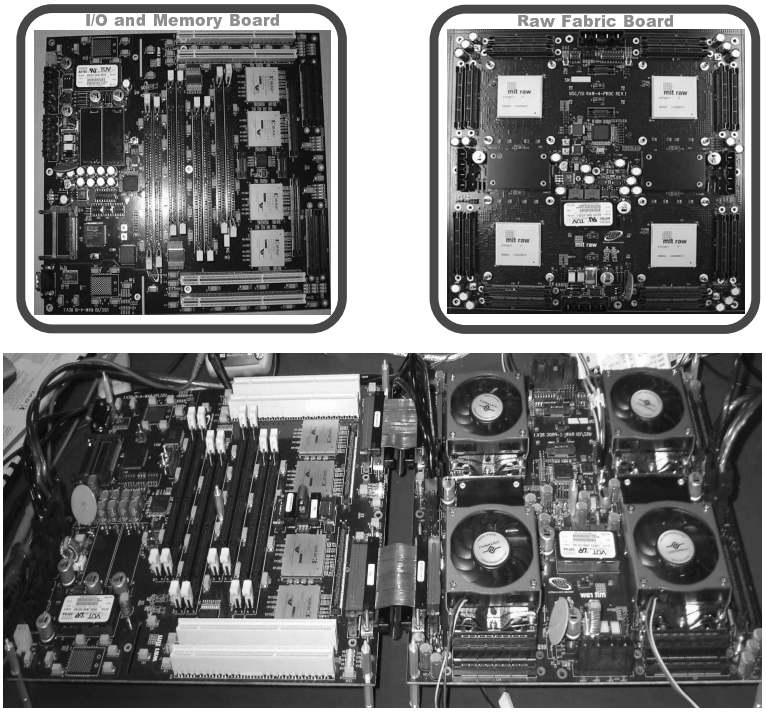

Figure 1: Photo of the Raw fabric and I/O boards.

The Raw fabric is a universal computational substrate suitable for signal processing and embedded applications. The key innovation behind Raw fabrics is the ability for software

to customize chip-level communication channels in an application-specific manner, thereby enabling the construction of mission-specific embedded systems cost-effectively. Raw fabrics offer the promise of orders of magnitude improvements for embedded applications when compared to microprocessor-based systems. These improvements are in performance, power, and size, and will allow system customization to be measured in hours instead of years. Raw Fabrics comprise single Raw chips with on-chip customizable interconnect, and board-level systems containing many Raw chips. Our project previously built a Raw chip prototype and a handheld computer system based on Raw. We have also designed and built a Raw fabric processor and I/O boards that make larger fabric systems easily realizable. We have built a 4 Raw chip fabric system, which is scalable to a 64 chip system that offers a one thousand processor system. Our results demonstrate that Raw performs at or close to the level of the best specialized machine for many application classes. When compared to a Pentium-III implemented in the same technology, Raw displays one to two orders of magnitude more performance for stream applications, while performing within a factor of two for sequential-desktop applications.

In addition to the Raw fabric hardware, we have also developed StreamIt: a language and compiler specifically designed for high performance, high productive stream computing. StreamIt raises the abstraction level in the streaming domain and provides a natural, high-level syntax that conceals architectural details without sacrificing performance. The StreamIt compiler is geared toward the application of novel stream-specific optimizations that achieve performance-levels comparable to expert programmers. We have publicly released the StreamIt compiler infrastructure to accelerate the StreamIt language adoption and application development. The release also includes a source-code distribution to promote compiler optimizations and innovations in the user community. The StreamIt compiler includes a wide array of optimizations for parallel architecture targets. The optimizations include load balancing, application layout, routing, and communication scheduling, as well as several automated domain-specific optimizations (e.g., translation of linear computation to the frequency domain) that are traditionally applied by domain-experts. We continue to

innovate our compilation technology with a focus on scalable and massively parallel PCA computing systems exemplified by the Raw fabric.

2 Overview of the Project

The Raw fabric [1] is a universal computational substrate that is suitable for signal processing and embedded applications. The key innovation behind raw fabrics is the ability for software to customize chip-level communication channels in an application-specific manner. Raw Fabrics comprise single chip Raw systems with on-chip customizable interconnect, and board-level systems that comprise one to many Raw chips.

Raw Fabrics address a major problem both with extant special purpose hardware systems and general purpose machines. First, modern supercomputers, built from state-of-the-art COTS microprocessors, have failed to eliminate the need for specialized hardware in the signal processing and embedded domains. Although supercomputing systems have the highest computational power, their inability to cost- effectively utilize this power for solving signal processing problems have led to the proliferation of ASICs, FPGAs, DSPs and full-custom hardware. Second, the need for special purpose hardware is even more acute in the embedded application domain, where efficient utilization of area, weight, and power is paramount. Unfortunately, the enormous cost and lengthy time-to-deployment of special-purpose hardware systems significantly reduces their appeal.

The Raw fabric draws its design motivation from both the strengths and weaknesses of using custom hardware for signal processing and embedded applications. There are three main benefits to developing custom hardware. First, when processing a stream of data, the ability to customize pipeline stages provides an order-of-magnitude performance improvement over a fixed pipeline when the energy budget is fixed. Second, custom hardware is able to efficiently orchestrate direct data movement between pipeline stages. In contrast, using a fixed memory hierarchy with caches is very inefficient in handling certain access patterns, such as stream data. Third, a custom design can tailor its resources to match both the level

and the granularity of the available parallelism in the application. This approach is more efficient than using a processor supporting a fixed amount of parallelism. A fourth, and minor advantage of custom hardware, is the ability to efficiently meet the granularity of data required by the application by customizing the size of registers, data paths, and ALUs.

Despite these advantages, custom hardware has a series of shortcomings. One of the biggest drawbacks in using custom hardware is its inflexibility. The inability to change the applications that run on a given hardware platform dramatically reduces their cost-effectiveness. Although FPGAs were partially successful in addressing this problem, seamless reconfiguration during continuous operation is yet to be achieved. More importantly, the inability of these devices to present an abstraction of unlimited resources renders the task of mapping programs to these devices incredibly difficult. Because ASICs are application-specific and cannot be applied to multiple problems, designing a custom ASIC for an algorithm can only afford a fraction of the development cost of designing a microprocessor. Therefore, it is not feasible to produce an ASIC with the same clock speed as a microprocessor of the same generation. Furthermore, it becomes prohibitively expensive to design a custom ASIC for each new process generation, while porting an application to a later version of a microprocessor is relatively simple.

Our project consisted of four major components that together provide a complete polymorphous computing environment. The four components are the Raw Processor, the Raw Fabric, the Raw Compiler, and the Raw Operating System. As described shortly, in each of the components, we resolved many open research issues and technical challenges.

In summary, this project designed and built a flexible and scalable computation fabric that can be morphed into solving many embedded applications in an energy, area, and time-efficient manner. A major component of this research was the Raw processor. The Raw processor is a simple tiled architecture with an innovative communication subsystem and is an ideal building block for larger computation fabrics. As such, the Raw fabric is an early proof-of-concept prototype of the general polymorphous computing architecture concept.

The project also completed the design of a multi-Raw-chip fabric. We have designed a 4 Raw chip fabric board along with an I/O board to demonstrate the ease of scalability of our design and philosophy. The boards can be tiled to build larger fabrics, and we have designed a methodology to massively scale the fabric, with a one thousand processor system planned in the future.

We also demonstrated in this project that stream computing presents unique opportunities to improve programmer productivity without sacrificing performance. We developed the StreamIt programming language and showed that many application with streaming data patterns are naturally expressed in the language. We also showed the StreamIt language features allow for novel optimizations that automate high-impact domain specific optimizations as well as machine specific optimizations. The StreamIt compiler which was developed during this project performs automatic discovery of concurrency, load balancing, communication and synchronization optimizations, leading to scalable performance on the Raw fabric system for many applications drawn from DSP (digital signal processing), well known and widely used Matlab codes, and the MIT Lincoln Laboratory benchmarks.

The project also investigated many novel microarchitectural features, language issues, compiler algorithms, and operating system components. Each of these is central to a successful polymorphous computing environment. The project developed several prototype Raw systems that are now in use at two DARPA sites including ISI and ATL at Lockheed Martin. Additional boards are in the process of being tested and they will go to several more of our DARPA collaborators.

3 Summary of Accomplishments and Systems Built

The following are the major components of our system that were implemented in our project.

1. We designed and successfully built a functional single-board Raw system. This was a key milestone of the project. The Raw processor has 16 processing tiles organized in

a 4x4 2-D mesh. We implemented the prototype Raw microprocessor in the SA-27E ASIC flow, which uses IBM's CMOS 7SF, a 180nm, 6-layer copper process. After a huge effort in the design and implementation using the IBM ASIC tools, as well has hand placement to achieve better clock speed and a huge effort on validation including multiple simulators, RTL level emulator, and countless tests, the fist silicon worked without a single bug! Although the IBM tools only projected the processor frequency to be 225 MHz, processor core ran at the frequency of 425MHz at 1.8V and 500 MHz at 2.2V. A 16-tile Raw processor has a throughput of 6.8GFLOPs.

2. We designed and implemented the Raw fabric board and the Raw fabric I/O board as well (along with our collaborators at ISI). The fabric board contains 4 Raw chips and can be connected in a mesh along with other fabric boards. The I/O board plugs into the periphery of the fabric board mesh and provides I/O, memory and other expansion functions. The boards provide a straightforward approach to building large scale fabrics, such as the planned 1K-processor fabric.

3. We developed an analytical framework that calculates the communication energy for point-to-point interconnection networks assuming various traffic/communication patterns. Using our model we compared the energy dissipation of operand communication for point-to-point networks against bus-based systems and presented a thorough analysis of the energy savings and advantages of point-to-point interconnection networks.

4. We investigated the class of streaming algorithms that can asymptotically reach the peak performance of the Raw processor. We demonstrated scalability of stream algorithms by running stream algorithms on simulated Raw fabrics of up to 1024 tiles. Our performance on the 1024 tile Raw fabric ranged from 414 GFLOPS for Matrix Multiplication to 294 GFLOPS for QR Factorization.

5. We built a complete steaming compiler and language called StreamIt. The language syntax is natural to most programmers of streaming systems, and allows the user to express stream programs effectively. StreamIt and the compiler are available to the

user community.

6. We engineered a wide array of StreamIt compiler optimizations for parallel architecture targets. The optimizations include load balancing, application layout, routing, and communication scheduling, as well as several automated domain-specific optimizations including translation of linear computation to the frequency domain, and a new class of state-space optimizations.

7. We extended the StreamIt compile to support a novel concept of compiler orchestrated space-time multiplexing of streams. This allows for the efficient execution of dynamic streams.

8. We developed a new unified and automatic methodology for extracting task-level, data-level, and pipeline parallelism from stream computations.

9. We showed that coarse-grained parallelism in stream programs provides robust and scalable performance for tiled architectures. Compared to baseline techniques for task and data level parallelization, we showed that our new techniques leads to 11x speedup for a 16-tile Raw processor for our benchmark suite.

10. We have designed and implemented the StreamIt Development Tool (SDT). We have fully integrated it with the Eclipse Universal Tools Platform. The StreamIt Debugger is a central component of the graphical SDT as it affords a practical methodology for reasoning about the correctness of largely parallel applications. The SDT also includes features that enable the editing and compilation of StreamIt programs using Eclipse. The SDT remains under active development, with a focus on scalability and visualization.

11. We have implemented, analyzed and distributed a new PCA benchmark suite called Versabench, and a new PCA performance metric called Versatility.

12. We have revised the StreamIt language to broaden the class of applications that can be naturally expressed. New applications domains include security codes, network

processing, video codecs, and multimedia.

13. We have designed and implemented an MPEG-2 codec in StreamIt. We also implemented a StreamIt version of the MIT Lincoln Labs Scalable Synthetic Compact Applications benchmark no. 3: Synthetic Aperture Radar, and showed that the benchmark has a straightforward translation from its Matlab equivalent.

14. We completed the implementation of the latest version of rMPI, and extensively tested its functionality, thereby allowing programmers to run large MPI applications on the Raw hardware. We have partially conducted a performance analysis of rMPI.

15. We designed, implemented, and optimized a software instruction cache. The scheme allows large programs to run on the Raw system. The overhead versus a traditional hardware cache is less than 30% on average and less than 10% on many benchmarks.

16. We developed Reptile, a port of the Trimaran ILP compiler to support the Raw architecture. Reptile extends the suite of available ILP optimization to include profile guided optimizations to improve the performance of irregular sequential code.

17. We built RawCC, which takes sequential C or FORTRAN programs and compiles them on to the Raw fabric. We developed the analysis necessary to extract ILP (instruction-level parallelism) out of sequential programs. Thus, programs written FORTRAN and C are able to use our compiler.

18. We conducted an educational workshop of the Raw system. About 20 researchers, current and potential users, attended the workshop. All the material and the video of the workshop are available from the Raw webpage.

19. We extended the Raw simulator to explore architectural extensions for future generations. The simulator supports different network configurations, an in order superscalar tile processor and different memory systems. Our simulator is calibrated against the real hardware.

20. We designed and implemented an MPEG-2 encoder for the Raw board and investigated performance bottlenecks. We implemented several optimizations for motion estimation and bit-rate control. We also developed a live-demo for an on-the-fly MPEG-2 encoder on Raw that runs on the Raw handheld board.

21. We designed and implemented an embedded wireless processor using a 2-pass system. The optimized multi-pass system achieved performance on par with equivalent non-multi-pass system implementations.

22. We designed and implemented an embedded networking fabric and control plane.

23. We designed and implemented a 32-node acoustic beamformer which was then extended to a record setting 1K nodes. This accomplishment was documented in the Guinness World Book of Records.

24. We designed and implemented the SUDS system as a hybrid compiler and runtime system that attempts to extract performance out of integer and sparse matrix applications where there is little parallelism and no compile time information. SUDS uses a part of the processor to implement a transaction memory system, this is able to speculatively parallelize and execute the program. SUDS is able to extract dynamic parallelism from applications and get comparable performance to a modern superscalar.

25. We developed a prototype host-based operating system for the Raw fabric. Our initial fabric operating system includes features needed to support I/O devices and OS services expected by applications. We developed and deployed a nano-kernel for each tile. A set of POSIX commands have been implemented using a few dedicated OS tiles and cooperating nano-kernels.

4 Accomplishments and Progress

We built a working prototype of a Polymorphous Computing Architecture platform based on the Raw infrastructure. The following sections summarize our findings and discuss the

components in detail.

4.1 Findings

We have shown that a complex processor with more than 100 million transistors can be designed, developed and validated at a university resulting in a chip without a single fault and that exceeds the expected clock speed. Our design produced the tightest coupling between two different pipelines in a chip multiprocessor. The operand can be moved from one pipeline to the next within three clock cycles. For the first time, we showed that exposing the wires to the compiler can drastically increase the processor utilization for applications that are statically analyzable.

We also demonstrated that the Raw tile abstraction is power efficient, and that using software functionality for functions traditionally relegated to hardware is not only practical but also contributes to the power savings. Specifically, our software instruction caching system is competitive against hardware instruction caches in terms of performance, while affording greater energy efficiency.

We showed that a class of algorithms called Stream algorithms, can be mapped on to scalable Raw processors where as the number of tiles of the processor is increased the algorithm reaches near peak processor utilization. We also showed that StreamIt applications benefit from Raw's exploitation of parallel resources and management of wires. The abundant parallelism and regular communication patterns in stream programs are an ideal match for the parallelism and tightly orchestrated communication on Raw. As stream programs often require high bandwidth, register-mapped communication serves to avoid costly memory accesses. Also, autonomous streaming components can manage their local state in Raw's distributed data caches and register banks, thereby improving locality. These aspects are key to the scalability demonstrated in the StreamIt benchmarks.

We demonstrated that we can automate tedious manual DSP optimizations using novel program analysis and optimizations that include linear dataflow analysis, linear combination,

frequency translation and automated optimization selection. These improve the performance of DSP programs written in the StreamIt language by an average factor of 8.

4.2 The Multi-Chip Raw Fabric

We designed and implemented the Raw Fabric system containing an array of Raw chips and I/O boards (along with our collaborators at ISI). The Raw Fabric system comprises the Raw fabric array board and the Raw fabric I/O board as well. The fabric board contains 4 Raw chips and can be connected in a mesh along with other fabric boards. The I/O board plugs into the periphery of the fabric board mesh and provides I/O, memory and other expansion functions. Figure 2 shows our processor board in the center, surrounded by three I/O boards.

We have tested and revised these boards.

4.3 The Raw Chip

The Raw processor was a major system that we built in the first phase of our project. We implemented the prototype Raw microprocessor in the SA-27E ASIC flow, which uses IBM's CMOS 7SF, a 180nm, 6-layer copper process. We received 120 chips from IBM in October of 2002. We are pleased to report that there were no bugs in first silicon.

Figure 3 shows a micro-photograph of the Raw die. The 16-tile geometry of the chip can be clearly made out.

4.4 The Raw Handheld Board

We validated the Raw processor on a prototype mother board called the Raw handheld board. We built several such boards along with our collaborators at ISI. Boards are in use at ISI and ATL (Lockheed Martin). Each board contains the Raw chip, SDRAM chips, I/O interfaces and interface FPGAs.

Figure 4 shows a photograph of the Raw motherboard.

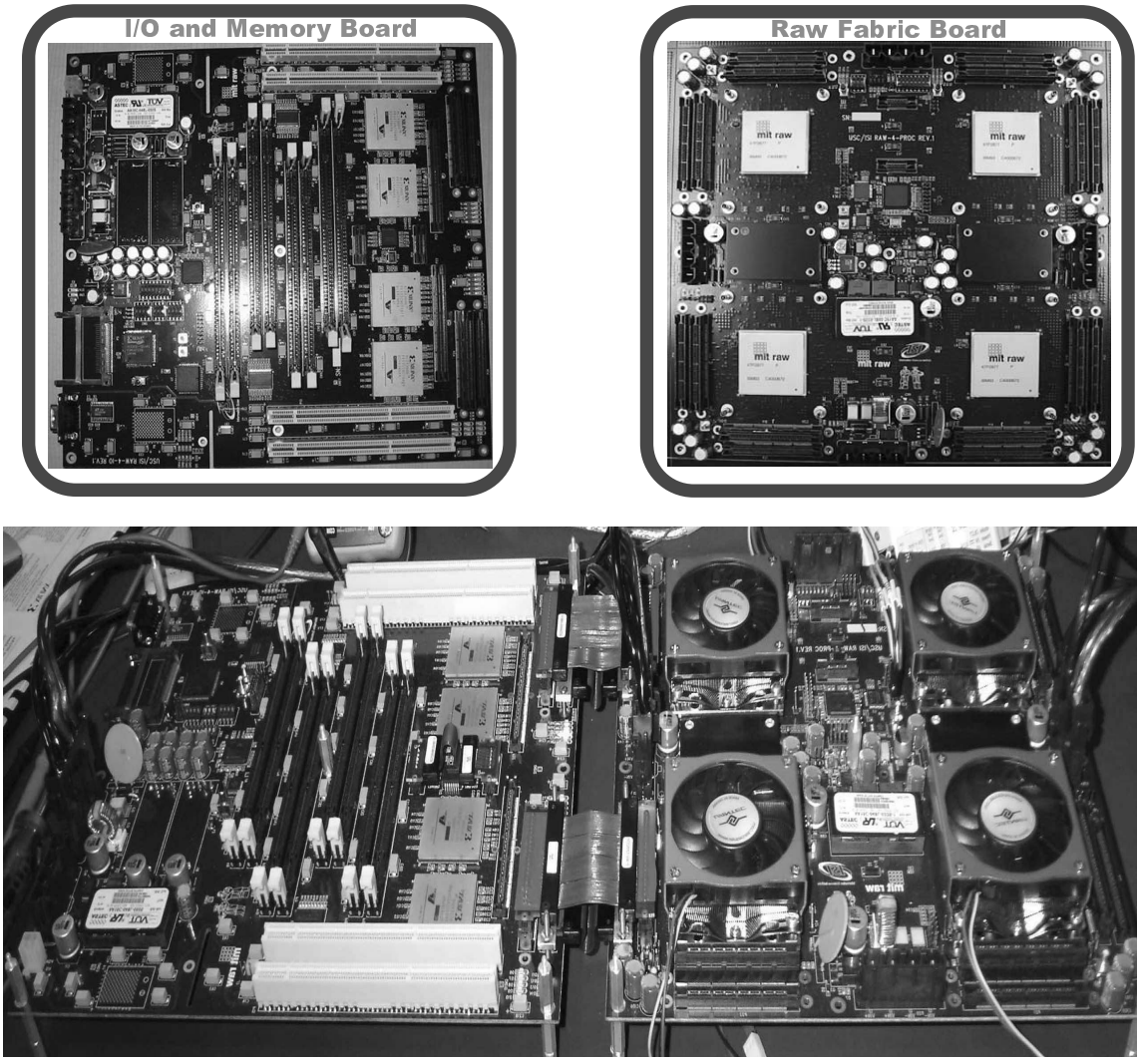

Figure 2: Photo of the Raw fabric and I/O boards.

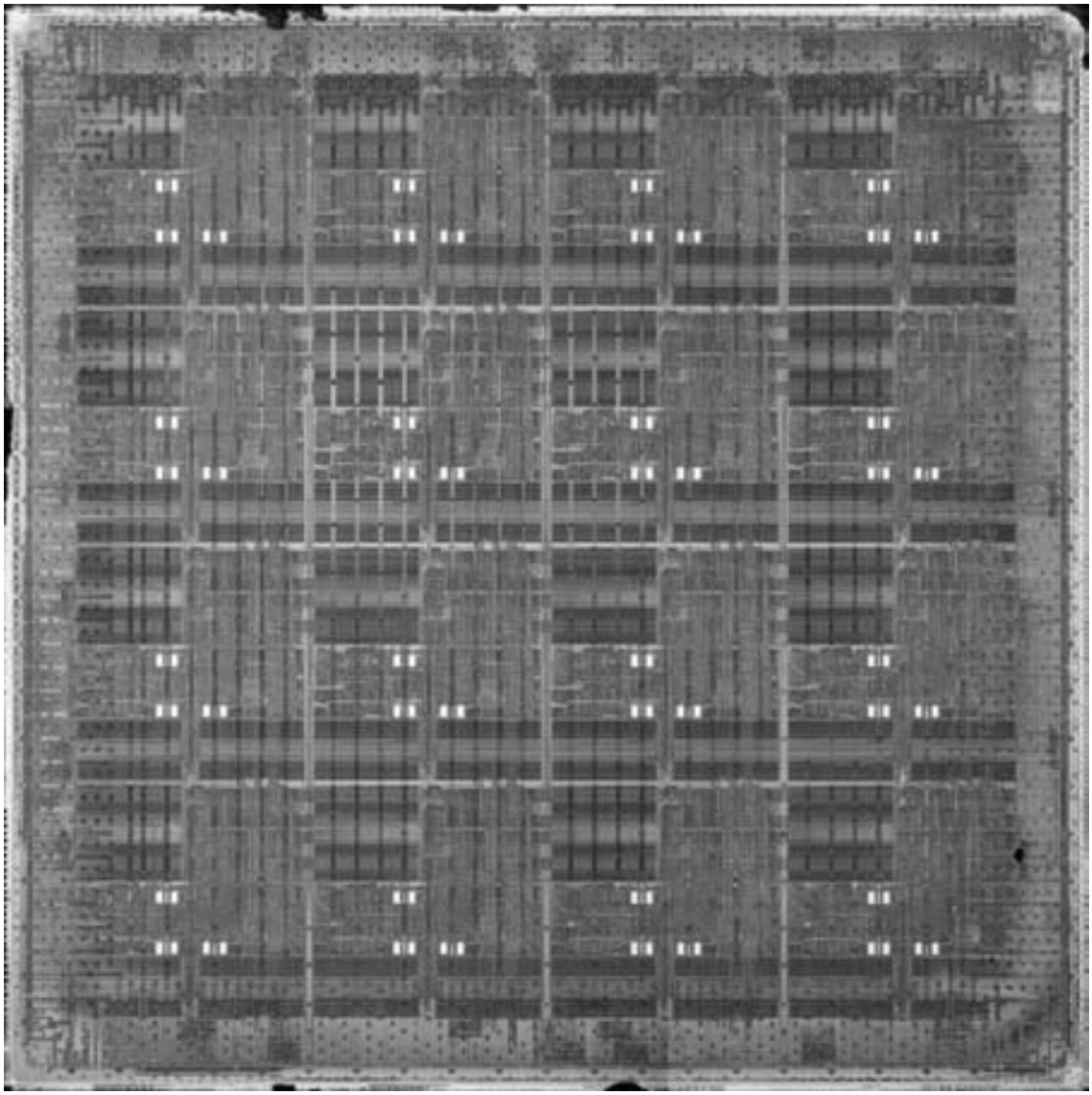

Figure 3: Photo of the Raw chip.

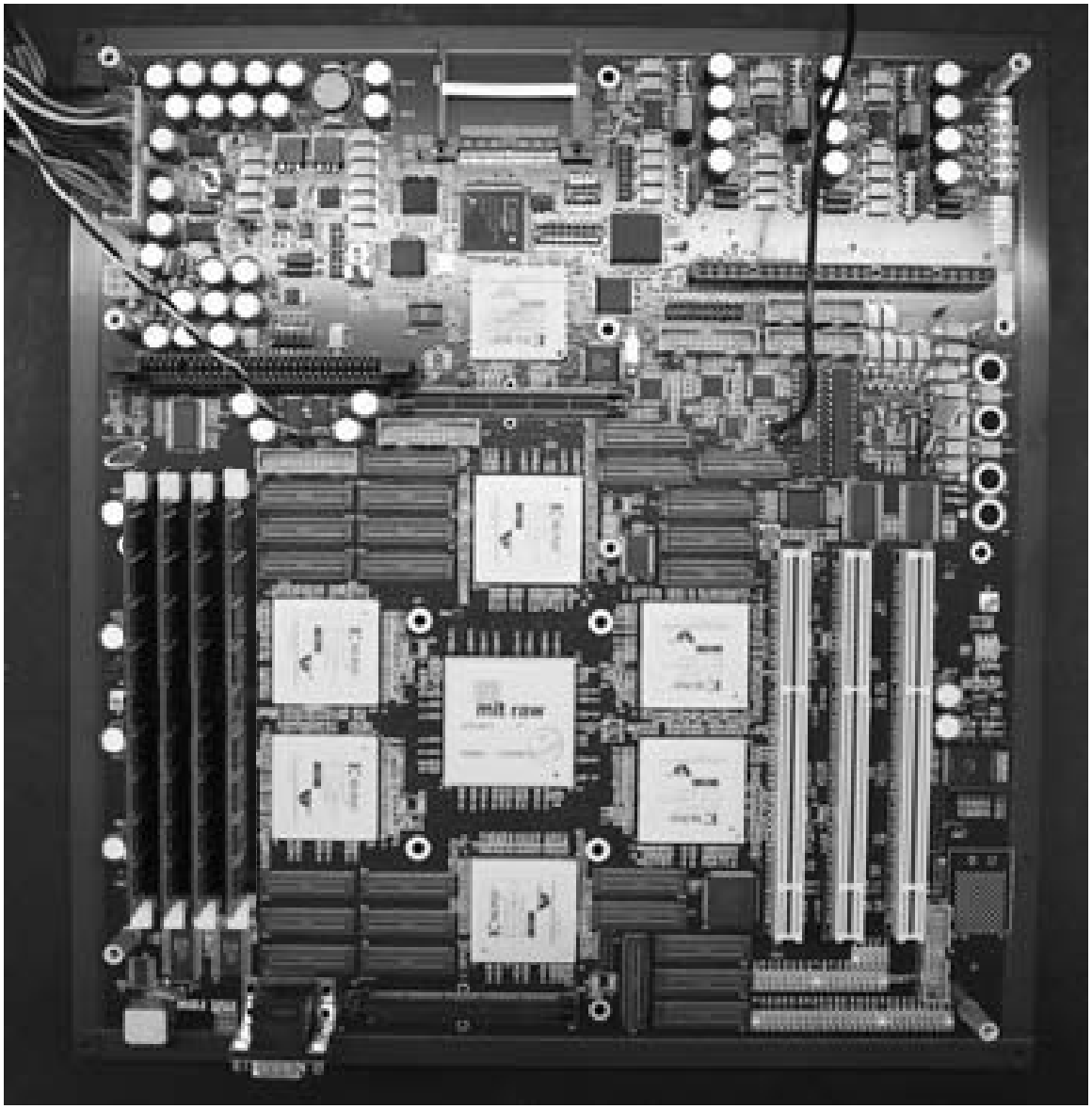

Figure 4: Photo of the Raw prototype motherboard. The board is 13 inches by 13 inches.

We have also designed an embedded wireless processor, an embedded networking fabric and control plane, and a 32-node acoustic beamformer (to be extended to 1K nodes in the following year).

We also implemented a high-speed USB interface for the motherboard. This allows the board to be connected to any laptop of PC with a USB2 interface.

We are nearing completion of the PCI interface, which will allow the Raw motherboard to function as a standalone system and provide even higher speed I/O.

4.5 The StreamIt Language

The StreamIt programming model allows the programmer to build an application by connecting components together into a stream graph, where the nodes represent filters that transform the data communicated along the edges. In StreamIt, the programmer is relieved of the burden of explicit buffer management and complex index expressions for multi-dimensional data. StreamIt also exposes the inherent parallelism and communication topology of the application, thereby empowering the compiler to perform many stream-aware optimizations that elude other languages. The end result is a clean, malleable, portable, and efficient code.

We used StreamIt to implement representative DSP, multimedia, and MATLAB applications. We compared our StreamIt code to other language such as C (commonly used for DSP and multimedia codes) and MATLAB (commonly used for mathematical codes). We showed that StreamIt is superior to C in exposing parallelism, and further more that the parallelism is exposed in very naturally ways to the programmers. We also showed that StreamIt code is far more malleable, and is more amenable to scalable parallelism than C and similar imperative languages. In StreamIt, the natural data (stream) flow is not obscured, and there is often a direct correlation between the block level diagram describing the flow of data between computation elements and the application syntax in StreamIt.

We also introduced a novel concept of teleport messaging. It allows for out-of-band communication of control parameters, allows programs to decouple the regular flow of data from

the irregular communication of parameters. This in turns leads to a cleaner implementation that is easier to maintain and evolve with changing software specifications.

As computer architectures change from the traditional monolithic processors, to scalable wire-exposed architecture like Raw, there will be a greater need for portable code implementations that expose parallelism and communication to enable efficient and high performance executions—while also boosting programmer productivity. StreamIt represents a step toward this end by providing a language that features hierarchical, modular, malleable, and portable streams.

4.6 The StreamIt StreamIt Compiler

We also built a complete steaming compiler for the StreamIt language. As noted above, StreamIt is a high-level, architecture-independent language for high-performance streaming applications. StreamIt contains language constructs that improve programmer productivity for streaming, including hierarchical structured streams, graph parameterization, and circular buffer management; these constructs also expose information to the compiler and enable novel optimizations. We developed a Raw backend for the StreamIt compiler, which includes fully automatic load balancing, graph layout, communication scheduling, and routing. StreamIt has been distributed to the DARPA community and the public via our website.

4.7 The RawCC Compiler

We built RawCC, which takes sequential C or FORTRAN programs and compiles them on to the Raw fabric. We developed the analysis necessary to extract ILP (instruction-level parallelism) out of sequential programs. Thus, programs written in the SUIF (Stanford University Intermediate Form) supported languages of FORTRAN, C, are able to use our compiler.

4.8 The Raw OS

We developed a prototype host-based operating system for the Raw fabric. Our initial fabric operating system includes features needed to support I/O devices and OS services expected by applications. We developed and deployed a nano-kernel for each tile. A set of POSIX commands have been implemented using a few dedicated OS tiles and cooperating nano-kernels. We have distributed this OS to the DARPA community.

We also developed RawGDB a debugger for Raw. RawGDB has also been released to external users.

We also implemented the compiler and runtime system needed for the software supported instruction cache and SUDS [2] (software undo system) systems.

4.9 Applications, Experimentation and Evaluation

We performed a substantial amount of experimentation of applications using the real Raw system. We also validated our simulator against the real hardware and conducted more experiments. We used our working RawCC compiler, the stream compiler and Raw OS for this task. ISI and Lincoln Labs have also developed several PCA applications on Raw.

Specifically, here are some highlights. The domains we examined include ILP computation, and stream and embedded computation. The performance of Raw in these individual areas are presented as comparison to a reference 600 MHz Pentium III, because the Pentium III was manufactured using the same 180 nm technology as Raw. We used a pair of 600 MHz Dell Precision 410 to run our reference benchmarks. These machines were outfitted with identical 100 MHz 2-2-2 PC100 256 MB DRAMs, and several microbenchmarks were used to verify that the memory system timings matched.

To compare the Raw and Dell systems more equally, the Raw simulator implemented a cycle-matched PC100 DRAM model and a chipset. This model has the same wall-clock latency and bandwidth as the Dell 410. However, since Raw runs at a slower frequency than

the P3, the latency, measured in cycles, is less. The term **RawPC** is used to describe a simulation which uses 8 PC100 DRAMs, occupying 4 ports on the left hand side of the chip, and 4 on the right hand side.

Because Raw is designed for streaming applications, we measured applications that use the full pin bandwidth of the chip. In this case, a simulation of CL2 PC 3500 DDR DRAM, which provides enough bandwidth to saturate both directions of a Raw port, was used. This is achieved by attaching 16 PC 3500 DRAMs to all 16 logical ports on the chip, in conjunction with a memory controller, implemented in the chipset, that supports a number of stream requests. A Raw tile can send a message over the general dynamic network to the chipset to initiate large bulk transfers from the DRAMs into and out of the static network. Simple interleaving and striding is supported, subject to the underlying access and timing constraints of the DRAM. This configuration is called **RawStreams**.

We note that Raw achieves greater than 16x speedup (versus a single tile) for several applications (listed in 6). Table 1 discusses the various factors that helped Raw.

Factor responsible	**Maximum Speedup**
Tile parallelism (Exploitation of Gates)	16x
Load/store elimination (Management of Wires)	3x
Streaming mode vs cache thrashing (Management of Wires)	60x
Streaming I/O bandwidth (Management of Pins)	60x
Increased cache/register size (Exploitation of Gates)	∼2x
Bit Manipulation Instructions (Specialization)	3x

Table 1: Sources of speedup for Raw over P3 (as configured in Table 3).

Tables 2 and 3 show functional unit timings and memory system characteristics for both systems, respectively. Table 4 shows Raw's measured power consumption. Table 5 lists a breakdown of the end-to-end message latency on Raw's scalar operand network. The low 3-

cycle inter-tile ALU-to-ALU latency and zero cycle send and receive occupancies are critical for obtaining good performance for ILP.

	Latency		Throughput	
Operation	**Raw Tile**	**P3**	**Raw**	**P3**
ALU	1	1	1	1
Load (hit)	3	3	1	1
Store (hit)	-	-	1	1
FP Add	4	3	1	1
FP Mul	4	5	1	1/2
Mul	2	4	1	1
Div	42	26	1	1
FP Div	10	18	1/10	1/18
SSE FP 4-Add	-	4	-	1/2
SSE FP 4-Mul	-	5	-	1/2
SSE FP 4-Div	-	36	-	1/36

Table 2: Functional unit timings on a single Raw tile and on a P3. Commonly executed instructions appear first. FP operations are single precision.

	1 Raw Tile	**P3**
CPU Frequency	425 MHz	600 MHz
Sustained Issue Width	1 in-order	3 out-of-order
Mispredict Penalty	3	10-15
DRAM Freq (RawPC)	100 MHz	100 MHz
DRAM Freq (RawStreams)	425 MHz	100 MHz
DRAM Access Width	8 bytes	8 bytes
L1 D cache size	32K	16K
L1 I cache size	32K	16K
L1 miss latency	54 cycles	7 cycles
L1 fill width	4 bytes	32 bytes
L1 line sizes	32 bytes	32 bytes
L1 associativities	2-way	4-way
L2 size	-	256K
L2 associativity	-	8-way
L2 miss latency	-	79 cycles
L2 fill width	-	8 bytes

Table 3: Memory system data for Raw tile and P3.

	Core	Pins
Idle - Full Chip	9.6 W	0.02 W
Average - Per Active Tile	0.54 W	-
Average - Per Active Port	-	0.2 W
Average - Full Chip	18.2 W	2.8 W

Table 4: Raw power consumption at 425 MHz, 25^o C

	Latency
Sending Processor Occupancy	0
Latency to Network Input	1
Latency per hop	1
Latency from Network Output to ALU	1
Receiving Processor Occupancy	0

Table 5: Breakdown of the end-to-end latency (in cycles) for a one-word message on Raw's static network.

Much like a VLIW (very long instruction word) architecture, Raw relies on the compiler to find and exploit ILP. We now examine how well Raw is able to support ILP. For this evaluation, we select a range of benchmarks that encompasses a wide spectrum of program types and degree of ILP. For some of the irregular integer benchmarks that RawCC is not mature enough to orchestrate, we compile and execute them on one tile to get a conservative worst case bound on their performance on Raw. Table 6 presents the performance of these benchmarks on RawPC and on the P3.

Of the benchmarks in our study, Raw is able to outperform the P3 for all the scientific benchmarks and several irregular applications. Of these, about half have speedups in the 2-3 range, but the other half have more promising speedups in the 4-7 range. At the other end of the spectrum for the integer applications run on a single Raw tile, our sampling of

applications showed that a Raw tile is roughly a factor of 2 slower.

Benchmark	Source	# Raw Tiles	Cycles on Raw	Speedup vs P3	
				by Cycles	by Time
Dense-Matrix Scientific Applications					
Swim	Spec95	16	58M	4.0	2.8
Tomcatv	Spec92	16	3.2M	1.9	1.4
Btrix	Nasa7:Spec92	16	4.6M	5.5	3.9
Cholesky	Nasa7:Spec92	16	5.5M	2.9	2.5
Mxm	Nasa7:Spec92	16	2.0M	3.5	2.5
Vpenta	Nasa7:Spec92	16	2.5M	10.3	7.3
Jacobi	Raw benchmark suite	16	150K	6.4	4.5
Life	Raw benchmark suite	16	4.0M	7.4	5.2
Sparse-Matrix/Integer Applications					
Fpppp-kernel	Spec92	16	150K	11.2	7.9
SHA	Perl Oasis	16	920K	1.9	1.3
Unstructured	CHAOS	16	15M	1.1	0.75
Adpcm	Mediabench	1	20M	0.85	0.60
GSM	Mediabench	1	310M	0.57	0.40
175.vpr	Spec 2000	1	2.9B	0.71	0.51
300.twolf	Spec 2000	1	2.3B	0.56	0.40

Table 6: Performance of sequential programs on Raw and on a P3.

Table 7 shows the speedups achieved by RawCC as the number of tiles varies from two to 16. The speedups are compared to performance of a single Raw tile. Overall, the source of speedups comes primarily from tile parallelism (see Table 1), but several of the dense matrix benchmarks benefit from increased cache capacity with parallel access as well (which explains the super-linear speedups). In addition, Fpppp-kernel benefits from increased register capacity, which leads to fewer spills.

	Number of tiles			
Benchmark	**2**	**4**	**8**	**16**
Dense-Matrix Scientific Applications				
Swim	1.4	2.2	4.3	7.9
Tomcatv	1.6	3.0	4.2	4.8
Btrix	1.6	4.5	10.3	21.8
Cholesky	1.9	3.7	6.2	6.1
Mxm	1.3	3.7	5.7	7.0
Vpenta	1.6	4.9	11.3	22.0
Jacobi	1.7	4.2	8.2	16.5
Life	0.8	2.0	4.9	10.5
Sparse-Matrix/Irregular Applications				
SHA	1.1	1.8	1.9	2.3
Fpppp-kernel	1.4	3.3	6.5	7.4
Unstructured	1.2	2.0	2.1	2.0

Table 7: Speedup of the ILP benchmarks relative to the single-tile Raw, from two to 16 tiles.

Next, we present performance of stream computations for Raw. Stream computations arise naturally out of real-time I/O applications as well as from embedded applications. The data sets for these applications are often large and may even be a continuous stream in real-time, which makes them unsuitable for traditional cache based memory systems. Raw

provides a more natural support for stream based computation by allowing data to be fetched efficiently through a register mapped, software orchestrated network.

The following results are for programs written in StreamIt, a high level stream language, and automatically compiled to Raw. We evaluate the performance of RawPC on several StreamIt benchmarks, which represent large and pervasive DSP applications. Table 8 summarizes the performance of 16 Raw tiles vs. a P3. For both architectures, we use StreamIt versions of the benchmarks; we do not compare to hand-coded C on the P3 because StreamIt performs at least 1-2X better for 5 of the 7 applications (this is due to aggressive unrolling and constant propagation in the StreamIt compiler). The comparison reflects two distinct influences: 1) the scaling of Raw performance as the number of tiles increases, and 2) the performance of a Raw tile vs. a P3 for the same StreamIt code. To distinguish between these influences, Table 9 shows detailed speedups relative to StreamIt code running on a 1-tile Raw configuration.

Benchmark	**Cycles Per Output on Raw**	**Speedup vs P3**	
		by Cycles	**by Time**
Beamformer	2675	6.6	4.6
Bitonic Sort	11	5.7	4.0
FFT	22	2.7	1.9
Filterbank	305	9.5	6.7
FIR	59	7.7	5.4
FMRadio	2610	9.6	6.8
Matrix Mult	183	5.4	3.8

Table 8: StreamIt performance results.

Benchmark	StreamIt on P3	StreamIt on n Raw tiles				
		1	2	4	8	16
Beamformer	2.6	1.0	3.7	4.0	8.1	17
Bitonic Sort	1.2	1.0	1.9	3.4	5.3	6.9
FFT	1.0	1.0	1.3	1.7	2.4	2.7
Filterbank	0.72	1.0	1.3	1.3	3.4	6.9
FIR	3.4	1.0	2.3	5.6	12	26
FMRadio	1.2	1.0	1.0	1.1	4.4	12
Matrix Mult	1.1	1.0	2.0	2.9	2.8	5.7

Table 9: Speedup (in cycles) of StreamIt benchmarks relative to a 1-tile Raw configuration. From left, the columns indicate the StreamIt version on a P3, and on Raw configurations with one to 16 tiles.

The primary result illustrated by Table 9 is that StreamIt applications scale effectively for increasing sizes of the Raw configuration. For FIR, FFT, and Bitonic, the scaling is approximately linear across all tile sizes (FIR is actually super-linear due to decreasing register pressure in larger configurations). For other applications, the scaling is slightly inhibited for small configurations. This is because 1) IMEM constraints prevent an unrolling optimization for small tile sizes (Beamformer, FM, Matrix Mult) and 2) there is some constant overhead that is amortized in larger configurations.

The second influence is the performance of a P3 vs. a single Raw tile on the same StreamIt code, as illustrated by the second column in Table 9. In most cases, performance is comparable. The P3 performs better in two cases because it can exploit ILP: Beamformer has independent real/imaginary updates in the inner loop, and FIR is a fully unrolled multiply-accumulate operation. In other cases, ILP is obscured by circular buffer accesses and control dependences.

In all, StreamIt applications benefit from Raw's exploitation of parallel resources and management of wires. The abundant parallelism and regular communication patterns in stream programs are an ideal match for the parallelism and tightly orchestrated communication on Raw. As stream programs often require high bandwidth, register-mapped communication serves to avoid costly memory accesses. Also, autonomous streaming components can manage their local state in Raw's distributed data caches and register banks, thereby improving locality. These aspects are key to the scalability demonstrated in the StreamIt benchmarks.

4.10 Support in Standardizing a Morphware Stable Interface

Our project has played an active role in the Morphware Forum.

Specifically, we have taken a leadership role in specifying the Streaming Virtual Machine (SVM) which will provide a common interface for high-level compilation tools to target all PCA architectures. In order to achieve high performance for streaming applications, the SVM

embraces a separation of control and data-intensive code, explicit communication via streams, and explicit memory management for streaming data. It also adopts a novel two-stage compilation strategy whereby high-level tools input a description of the target architecture (using the PCA Machine Model) and compile to a version of the SVM that is parameterized for that architecture. Reaching consensus on this interface has involved detailed design discussions with many of the PCA teams, including Stanford, Raytheon, Georgia Tech, USC and the University of Texas.

Along with Reservoir Labs, we have coordinated the design process and have produced a stable specification that is serving as a cornerstone of the Morphware toolchain. We have also expressed the Raw architecture in terms of the Machine Model, and are actively engaged with the evaluation of the Reservoir High-Level Compiler that targets the SVM.

We also implemented a compiler that translates StreamIt code to the Reservoir streaming language, and provided the Morphware forum with the most comprehensive results that compare native compilation of streaming code to Raw versus the two stage compilation approach adopted by the forum. Our results which were gathered in collaboration with ISI serve to identify inefficiencies with the current two-stage compilation methodology.

5 Conclusions and Recommendations for Future Work

This report described the architecture and implementation of the Raw prototype. We have shown that a complex processor with more than 100 million transistors can be designed, developed, and validated at a university resulting in a chip without a single fault and that exceeds the expected clock speed. The Raw processor has 16 processing tiles organized in a 4x4 2-D mesh, and a throughput of 6.8GFLOPs at 425MHz.

Our work has demonstrated that Raw's exposed ISA (instruction set architecture) allows parallel applications to exploit all of the chip resources, including gates, wires and pins. Raw supports ILP by scheduling operands over a scalar operand network that offers very

low latency for scalar data transport. Raw's compiler manages the effect of wire delays by orchestrating both scalar and stream data transport. The Raw processor demonstrates that existing architectural abstractions like interrupts, caches, and context-switching can continue to be supported in this environment, even as applications take advantage of the low-latency scalar operand network and the large number of ALUs.

Our results demonstrate that the Raw processor performs at or close to the level of the best specialized machine for each application class. When compared to a Pentium III, Raw displays one to two orders of magnitude more performance for stream applications, while performing within a factor of two for low-ILP applications. It is our hope that the Raw research will provide insight for architects who are looking for ways to build versatile processors that leverage the vast silicon resources while mitigating the considerable wire delays that loom on the horizon.

Our effort has pointed to several future directions that are worth exploring: (1) evaluating the performance for much larger numbers of tiles and a wider set of programs, (2) generalizing on-chip operand networks so that they support other forms of parallelism exploited by microprocessors such as stream parallelism and thread parallelism, (3) complete designs and evaluation of both dynamic and compile-time schemes for operation/operand assignment and scheduling, (4) mechanisms for fast exception handling and context switching, (5) a thorough analysis of the tradeoffs between commit point, exception handling capability, and network latency, (6) low energy tiled processors and scalar operand networks, and (7) tiled architectures with support for bit-level processing.

We introduced StreamIt as a language and compiler specifically designed for modern stream programming. StreamIt raises the abstraction level in the streaming domain and provides a natural, high-level syntax that conceals architectural details without sacrificing performance. We publicly released the StreamIt compiler infrastructure to accelerate the StreamIt language adoption and application development. The release also includes a source-code distribution to promote compiler optimizations and innovations in the user community.

We demonstrated that automating the design process of a DSP engineer by introducing novel compiler analysis and optimizations, linear dataflow analysis, linear combination, frequency translation and automated optimization selection, we can improve the performance of DSP programs written in StreamIt by an average factor of 8. We released our collection of streaming applications as part of the StreamIt Benchmarks Suite. The benchmarks serve as a standardized mechanism for the evaluation of streaming optimizations and architectures.

6 Key Publications

The following references list our major publications: [3], [1], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26].

References

[1] Michael Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Jae-Wook Lee, Paul Johnson, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, pages 25–35, March/April 2002.

[2] Matthew Frank, C. Andras Moritz, Benjamin Greenwald, Saman Amarasinghe, and Anant Agarwal. SUDS: Primitive Mechanisms for Memory Dependence Speculation. Technical report, M.I.T., January 6 1999.

[3] C. Andras Moritz, Donald Yeung, and Anant Agarwal. SimpleFit: A Framework for Analyzing Design Tradeoffs in Raw Architectures. *IEEE Transactions on Parallel and Distributed Systems*, July 2001.

[4] Volker Strumpen, Henry Hoffmann, and Anant Agarwal. A Stream Algorithm for the SVD. Technical Memo MIT-LCS-TM-641, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, October 2003.

[5] Henry Hoffmann, Volker Strumpen, and Anant Agarwal. Stream Algorithms and Architecture. Technical Memo MIT-LCS-TM-636, Laboratory for Computer Science, Massachusetts Institute of Technology, March 2003.

[6] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fataneh Ghodrat, Benjamin Greenwald, Henry Hoffmann, Paul Johnson, Walter Lee, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Saman Amarasinghe, and Anant Agarwal. A 16-Issue Multiple-Program-Counter Microprocessor with Point-To-Point Scalar Operand Network. In *Proceedings of the IEEE International Solid-State Circuits Conference*, 2003.

[7] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2003.

[8] Jason Kim, Michael Bedford Taylor, Jason Miller, and David Wentzlaff. Energy Characterization of a Tiled Architecture Processor with On-Chip Networks. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2003.

[9] Diego Puppin, Mark Stephenson, Saman Amarasinghe, Una-May O'Reilly, and Martin C. Martin. Adapting convergent scheduling using machine learning. In *Languages and Compilers for Parallel Computing*, College Station, TX, October 2003.

[10] Walter Lee, Diego Puppin, Shane Michael Swenson, and Saman Amarasinghe. Convergent scheduling. In *International Symposium on Microarchitecture*, Istanbul, Turkey, November 2002.

[11] Mark Stephenson, Johnathan Babb, and Saman Amarasinghe. Bitwidth analysis with application to silicon compilation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Vancouver, British Columbia, June 2000.

[12] Rajeev Barua, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Compiler support for scalable and efficient memory systems. *IEEE Transactions on Computers*, 50(11), November 2001.

[13] Gleb A. Chuvpilo, David Wentzlaff, and Saman Amarasinghe. Gigabit ip routing on raw. In *IEEE HPCA Workshop on Network Processors*, pages 2–9, Cambridge, Massachusetts, February 2002.

[14] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. In *International Conference on Compiler Construction*, Grenoble, France, April 2002.

[15] Michael Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Chris Leger, Andrew A. Lamb, Jeremy Wong, Henry Hoffman, David Z. Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA USA, October 2002.

[16] Michal Karczmarek, William Thies, and Saman Amarasinghe. Phased scheduling of stream programs. In *Languages, Compilers, and Tools for Embedded Systems*, San Diego, CA, June 2003.

[17] Andrew A. Lamb, William Thies, and Saman Amarasinghe. Linear analysis and optimization of stream programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, CA, June 2003.

[18] Gleb A. Chuvpilo and Saman Amarasinghe. High-bandwidth packet switching on the raw general-purpose architecture. In *International Conference on Parallel Processing*, Kaohsiung, Taiwan, Republic of China, October 2003.

[19] Janis Sermulins, William Thies, Rodric Rabbah, and Saman Amarasinghe. Cache aware optimization of stream programs. In *Languages, Compilers, and Tools for Embedded Systems*, Chicago, June 2005.

[20] William Thies, Michal Karczmarek, Janis Sermulins, Rodric Rabbah, and Saman Amarasinghe. Teleport messaging for distributed stream programs. In *Symposium on Principles and Practice of Parallel Programming*, Chicago, Illinois, June 2005.

[21] Jiawen Chen, Michael Gordon, William Thies, Matthias Zwicker, Kari Pulli, and Fredo Durand. A reconfigurable architecture for load-balanced rendering. In *Graphics Hardware*, Los Angeles, CA, August 2005.

[22] Sitij Agrawal, William Thies, and Saman Amarasinghe. Optimizing stream programs using linear state space analysis. In *Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, San Francisco, CA, September 2005.

[23] Michael Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Benjamin Greenwald, Henry Hoffman, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams. In *International Symposium on Computer Architecture*, Munich, Germany, June 2004.

[24] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar Operand Networks. *IEEE Transactions on Parallel and Distributed Systems*, February 2005.

[25] Matthew Drake, Henry Hoffman, Rodric Rabbah, and Saman Amarasinghe. Mpeg-2 decoding in a stream programming language. In *International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, April 2006.

[26] Michael Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *International Conference*

on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 2006.

DISTRIBUTION LIST

DTIC/OCP 8725 John J. Kingman Rd, Suite 0944 Ft Belvoir, VA 22060-6218	1 cy
AFRL/VSIL Kirtland AFB, NM 87117-5776	2 cys
AFRL/VSIH Kirtland AFB, NM 87117-5776	1 cy
Official Record Copy AFRL/VSSE, Capt Angelo Bonavita	1 cy